Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación IIC1103

Prof. Ignacio Casas
icasas@ing.puc.cl

Tema 06 – Programación Orientada a Objetos
Parte 1.2 Ejemplos

Colaboración de Mauricio Arriagada

Agenda

1. Modelación (Programación) con Objetos
2. Tipos de datos y Clases
3. Python: Creando Clases
4. Python: Creando Objetos de una Clase
5. Ejercicios

6. **Mini-Tarea 11:** Resolver el problema 1 de la Interrogación 2 del 2015-2.

Recordando: para crear objetos / instancias

- Para crear (instanciar) un objeto de una clase:

  variable = Nombre_Clase (param1, param2, …)

- Por ejemplo:

```
class Carta:
    """ atributos: pinta, valor
    """
    def __init__ (self, pinta, valor):
        self.pinta = pinta
        self.valor = valor

""" instanciando (creando) 2 objetos de la clase carta """
mi_carta_1 = Carta("Trebol", 1)
mi_carta_2 = Carta("Corazon", 2)
```

Los objetos en **mi_carta_1** **mi_carta_2** son instancias de la clase **Carta**.

3

Ejemplo 1: Modelemos un Banco Comercial

Cuentas Bancarias y Clientes-Personas

- Queremos modelar una cuenta bancaria que permita hacer retiros y depósitos de dinero.

Objeto: cuenta bancaria

Clase: todas las cuentas bancarias

Balance — **atributo**

retirar

depositar

métodos

4

Ejemplo 1: Banco (modelación con objetos)

Cuenta Bancaria

- **Objetos y Clases**

Ojo con el parámetro por defecto

```
class CuentaBancaria:
      """ Atributo: balance """
      def __init__(self, balance = 0):
                  self.balance = balance

      def retirar(self, cantidad):
                  self.balance -= cantidad
                  return self.balance

      def depositar(self, cantidad):
                  self.balance += cantidad
                  return self.balance
"""" Comienza programa principal """"
cuenta1 = CuentaBancaria()
cuenta2 = CuentaBancaria()
cuenta3 = CuentaBancaria(1000)
```

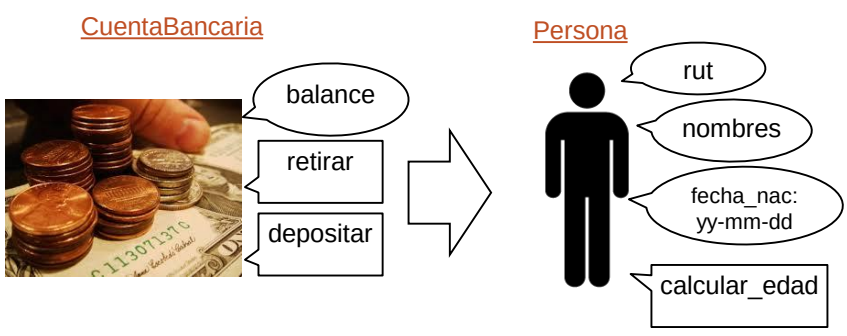
Estos métodos retornan un valor.

5

Ejemplo 1: Banco (programación orientada a objetos)

Cuenta Bancaria - Persona

- Ahora queremos asociar cada cuenta bancaria a una persona distinta.



6

Ejemplo 1: Modelación del Banco con objetos

Clase Persona

```
import datetime
class Persona:
    """ Atributos: rut, nombres, fnac
    """
    def __init__(self, rut="",nombres="",fnac=""):
        self.rut = rut
        self.nombres = nombres
        self.fnac = fnac
    def calcular_edad(self):
        y,m,d = self.fnac.split("-")
        now = datetime.datetime.now()
        return now.year - int(y)
```

Vimos anteriormente que **split** es un método de la clase string.

7

Ejemplo 1: Banco (orientación a objetos)

Cuenta Bancaria

- Comunicación entre los objetos CuentaBancaria y Persona

```
class CuentaBancaria:
    """ Atributos: balance, persona """
    def __init__(self):
        self.balance = 0

    def retirar(self, cantidad):
        self.balance -= cantidad
        return self.balance

    def depositar(self, cantidad):
        self.balance += cantidad
        return self.balance

    def asignar_titular(self,persona):
        self.persona = persona
```

Decidimos mejor iniciar todas las cuentas en 0.

Ojo que el atributo persona no existía. Ahora sí: se crea en este método.

Ejemplo 1: Banco (orientación a objetos)

Cuenta Bancaria asociada a una Persona

1 

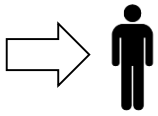
```
jp = Persona( '11222333-0','Juan Perez',‘1990-07-01’)
```

2 

```
cuenta_001 = CuentaBancaria()
```

3

```
cuenta_001.asignar_titular(jp)
```



9

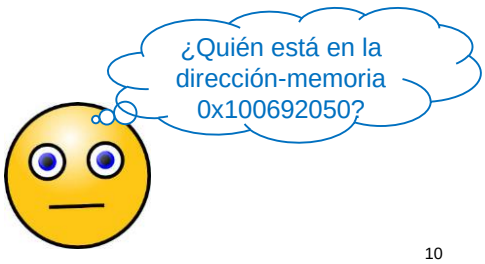
Ejemplo 1: Banco (orientación a objetos)

Cuenta Bancaria

Nos gustaría ahora imprimir el nombre del titular de una cuenta:

```
jp = Persona('11222333-0','Juan Perez','1990-07-01')
cuenta_001 = CuentaBancaria ()
cuenta_001.asignar_titular (jp)
print (cuenta_001.persona)
```

```
<__main__.Persona object at 0x100692050>
```



Pero lo que queremos es imprimir (obtener como un “string”) los atributos de una instancia de esta clase:

10

Ejemplo 1: Banco (orientación a objetos)
Cuenta Bancaria

Nos gustaría mostrar (imprimir) los atributos de un objeto de esta clase, por ejemplo, mostrar el siguiente mensaje:

Hola! soy Juan Perez y soy dueño de esta cuenta

Lo podemos hacer así:

```
jp = Persona("11222333-0","Juan Perez","1990-07-01")
cuenta_001 = CuentaBancaria()
cuenta_001.asignar_titular(jp)
print("Hola!, soy ",cuenta_001.persona.nombres,
      " y soy dueño de esta cuenta")
```

```
>>>
Hola!, soy  Juan Perez  y soy dueño de esta cuenta
```

Alternativamente, podemos utilizar el método especial **__str__** para "sobrecargar" la función **str**

11

Ejemplo 1: Banco (orientación a objetos)
Método especial __str__ llamado "sobrecarga" de la función str

Agregamos en la clase Persona una función especial **__str__** para que al momento de imprimir (print) un objeto de esa clase, muestre el mensaje que deseamos. Esto se llama "sobrecarga" de la función **str**.

```
import datetime
class Persona:
    """ Atributos: rut, nombres, fnac
    """
    def __init__(self, rut="",nombres="",fnac=""):
        self.rut = rut
        self.nombres = nombres
        self.fnac = fnac
    def calcular_edad(self):
        y,m,d = self.fnac.split("-")
        now = datetime.datetime.now()
        return now.year - int(y)
    def __str__ (self):
       return("Hola! soy "+str(self.nombres)+
              " y soy el dueño de esta cuenta")
```

Método especial que "sobrecarga" a la función **str**

12

Ejemplo 1: Banco (orientación a objetos)

Método especial __str__ llamado "sobrecarga" de la función str

Ahora podemos crear un objeto como instancia de **CuentaBancaria** con su titular e imprimirlo tal como queríamos:

```
# comienza programa principal
jp = Persona("11222333-0","Juan Perez","1990-07-01")
cuenta_001 = CuentaBancaria()
cuenta_001.asignar_titular(jp)
print(cuenta_001.persona)
```

El método **__str__** que definimos en la clase **Persona** permite imprimir directamente el objeto.

```
>>>
Hola! soy Juan Perez y soy el dueño de esta cuenta
```

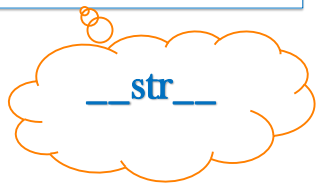
13

Ejemplo 1: Banco (orientación a objetos)

Método especial __str__ llamado "sobrecarga" de la función str

Ahora modifica el método **__str__** de la clase Persona para que al momento de imprimir (print) un objeto de esa clase, muestre el siguiente mensaje:

```
jp = Persona('11222333-0','Juan Perez','1990-07-01')
cuenta_001 = CuentaBancaria()
cuenta_001.asignar_titular(jp)
print(cuenta_001.persona)

>>>
Hola! soy Juan Perez y tengo 25 años
```



14

Ejemplo 1: Banco (orientación a objetos)

Método especial **__str__** llamado "sobrecarga" de la función **str**

```
class Persona:
        """ Atributos: rut, nombres, fnac
        """
    def __init__(self, rut, nombres, fnac):
        self.rut = rut
        self.nombres = nombres
        self.fnac = fnac

    def calcular_edad(self):
        y,m,d = self.fnac.split ( "-" )
        now = datetime.datetime.now()
        return now.year – int (y)

    def __str__(self):
        mensaje = "Hola! soy {0} y tengo {1} años"
        edad = self.calcular_edad()
        return  mensaje.format (self.nombres, edad)
```

Método format para la clase **str**

15

Cont. Ejemplo 1 Banco (orientación a objetos)

Continuemos con la Cuenta Bancaria

Una alternativa válida al método **asignar_titular** es definir en **CuentaBancaria** al objeto **persona** como atributo dentro de __**init**__

De esta forma, podremos asegurar que toda cuenta tenga un titular al ser creada.

Quedaría así:

16

Ejemplo 1 Banco (Orientación a objetos)

Cuenta Bancaria con titular persona al inicio

```
class CuentaBancaria:
      """ Atributos: balance, persona """
      def __init__(self, p):
            self.balance = 0  # se crea una cuenta inicial con 0 lucas
            self.persona = p  # copia el objeto p en persona

    def retirar(self, cantidad):
        self.balance -= cantidad
        return self.balance

    def depositar(self, cantidad):
        self.balance += cantidad
        return self.balance

    def __str__(self):
         plantilla= "Balance {0}"
        return plantilla.format(self.balance)
```

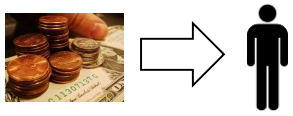
```
jp = Persona( '11222333-0','Juan Perez','1990-07-01')
cuenta_001 = CuentaBancaria(jp)
print ( jp + "y tengo mi " + cuenta_001 + " lucas" )
```

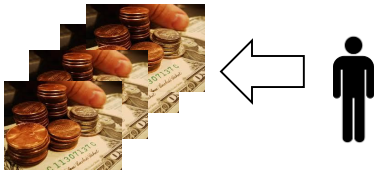
17

Ejemplo 1 Banco (orientación a objetos)

Personas : varias Cuentas bancarias

- En un banco una cuenta bancaria tiene un titular pero los clientes pueden tener una o varias cuentas bancarias.





18

Ejemplo 1 Banco (Orientación a objetos)

Persona : varias cuentas

```
class Persona:
      “”” Atributos: rut, nombres, fnac, cuentas (lista) “””

      def __init__(self, rut, nombres, nac):
            self.rut = rut
            self.nombres = nombres
            self.f_nac = nac
            self.cuentas = []

      def calcular_edad(self):
            y,m,d = self.fnac.split('-')
            now = datetime.datetime.now()
            return now.year - int(y)

      def __str__(self):
            plantilla = "Hola! soy {0} y tengo {1} años"
            edad = self.calcular_edad()
            return plantilla.format(self.nombres, edad)
```

Agregamos un nuevo atributo (una lista)

Ejemplo 1

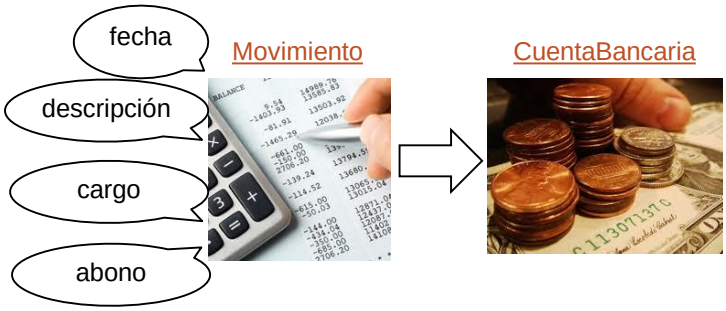
Persona : varias cuentas

```
jp = Persona('11222333-0','Juan Perez','1990-07-01')

jp.cuentas.append(CuentaBancaria(jp))
jp.cuentas.append(CuentaBancaria(jp))

jp.cuentas[0].depositar(1000)
jp.cuentas[1].retirar(2000)

for cuenta in jp.cuentas:
    print(cuenta.balance)

>> 1000
>> -2000
```

Ejemplo 1

Personas : Cuentas Bancarias

- Además, en un banco por seguridad, confianza y seguimiento (por ley) se registran los movimientos
  - (retiros y depósitos) de cada cuenta.

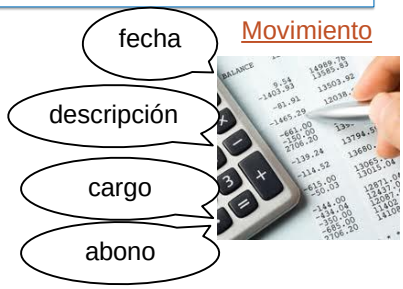


21

Ejemplo 1

Movimiento Bancario

```
import datetime
class Movimiento:
    """ Atributos: fecha, desc, cargo, abono  """
    def __init__(self, desc, cargo=0, abono=0):
        self.fecha = datetime.datetime.now()
        self.desc = desc
        self.cargo = cargo
        self.abono = abono
```



22

Ejemplo 1 Banco (Orientación a objetos)

Cuenta Bancaria con Movimientos

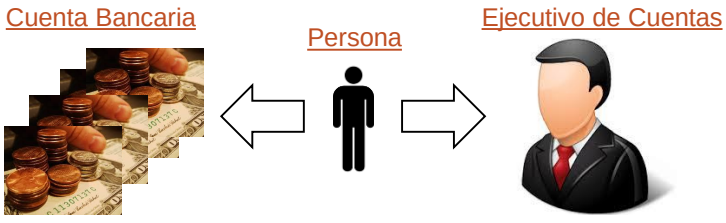
```
class CuentaBancaria:
    """ Atributos: balance, persona, movimientos """
    def __init__(self, persona):
        self.balance = 0
        self.persona = persona
        self.movimientos = []

    def retirar(self, cantidad):
        self.balance -= cantidad
        self.movimientos.append(Movimiento("Giro",cantidad,0))
        return self.balance

    def depositar(self, cantidad):
        self.balance += cantidad
        self.movimientos.append(Movimiento("Depósito",0,cantidad))
        return self.balance

    def __str__(self):
        plantilla = "Balance {0}"
        return plantilla.format(self.balance)
```

23

Ejemplo 1

Personas : Cuentas Bancarias : Ejecutivo de Cuentas

- Ahora el banco define una estrategia para fidelizar a sus clientes mediante atenciones personalizadas. Para ello decide asignar a cada cliente un ejecutivo de cuentas.



24

Ejemplo 1 Banco (agregando nuevas Clases de objetos)

Ejecutivo de Cuentas

Definimos una nueva clase en nuestro modelo:

```
class EjecutivoCuentas:
    """ Atributos: rut, nombres, clientes """
    def __init__(self, rut, nombres):
        self.rut = rut                  # rut del ejecutivo
        self.nombres = nombres          # nombres del ejecutivo
        self.clientes = []              # clientes del ejecutivo

    def agregar_cliente(self, persona):
        self.clientes.append(persona)
```

Y agregaremos un nuevo atributo "**ejecutivo**" en la clase **Persona**:

25

Ejemplo 1 Banco (Extendiendo el modelo)

Persona : varias cuentas, un ejecutivo

```
class Persona:
    """ Atributos: rut, nombres, fnac, cuentas (lista), ejecutivo (persona) """
    def __init__(self, rut, nombres, nac):
        self.rut = rut
        self.nombres = nombres
        self.f_nac = nac
        self.cuentas = []
        self.ejecutivo = ""

    def calcular_edad(self):
        y,m,d = self.fnac.split('-')
        now = datetime.datetime.now()
        return now.year - int(y)

    def __str__(self):
        plantilla = "Hola! soy {0} y tengo {1} años"
        edad = self.calcular_edad()
        return plantilla.format(self.nombres, edad)
```

Agregamos un nuevo atributo **ejecutivo**

26

Ejemplo 1

Cliente – Ejecutivo de Cuentas

```
jp = Persona (  '11222333-0' , 'Juan Perez ', '1990-07-01'  )

jp.cuentas.append (CuentaBancaria(jp))
jp.cuentas.append (CuentaBancaria(jp))

jp.ejecutivo = EjecutivoCuentas ( '123123', 'Lucho Farkas' )
```

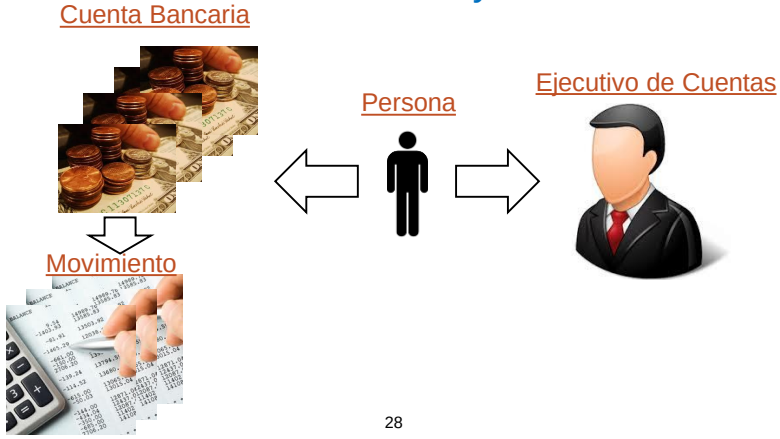
Entonces, podemos "extender – escalar" nuestro modelo agregando nuevos atributos, sin necesidad de re-definir la clase Persona.

27

Fin del Ejemplo 1 (la ventaja de la POO)

Personas : Cuentas Bancarias : Ejecutivo de Cuentas

- Utilizando POO podemos crear programas escalables, mantenibles y re-utilizables



28

Ejemplo 2: Juguemos 21Real (BlackJack)

El siguiente ejemplo utiliza objetos de la clase **listas** (clase pre-definida llamada **list**) que estudiamos en el capítulo anterior.

29

Ejemplo 2: Generemos un mazo de cartas (con listas)

Utilizaremos la definición de la clase Carta anterior y agreguemos un mazo de cartas (baraja inglesa) desordenadas :

import random

```
class Mazo:
    def __init__(self):
        self.mazo = []
        self.generar_mazo()

    def generar_mazo(self):
        for p in ['♥', '♦','♠','♣']:
            for i in range (2,11):
                carta = Carta(p,i)
                self.mazo.append(carta)

            for i in ['A','J','Q','K']:
                self.mazo.append(Carta(p,i))
        random.shuffle(self.mazo)
```

Para poder utilizar el método shuffle tenemos que "importar" la libr **random**.

La clase **Mazo** se define con un atributo **mazo** (una lista de objetos **Carta**) y un método **generar_mazo**

Ahora podemos crear un mazo con la instrucción:

mi_mazo = Mazo()

Ejemplo 2: Podemos "enriquecer" **Carta** con más métodos

Para poder jugar 21Real (21R, Black Jack) agregaremos otros métodos a la clase **Carta**:

```
class Carta:
    """ Atributos de instancias: pinta, valor
        Atributo de la Clase: mono_10 """
    mono_10 = ['K','Q','J']
    def __init__(self, pinta, valor):
        self.pinta = pinta
        self.valor = valor

    def __str__(self):
        return str(self.valor) + self.pinta

    def get_valor(self):
        if self.valor in self.mono_10:    # 'K','Q','J'
            return 10
        elif self.valor == "A":  # puede valer 1 u 11
            return 1             # lo dejámos en 1 por aho
        else:
            return self.valor
```

Agregamos el método **__str__** para poder imprimir fácilmente la pinta y valor de una carta cómo un string.

Agregámos el método get_valor para poder saber el valor (en 21R) de una carta.

""" Programa ppal """

```
mi_carta = Carta( '♠' , 'K' )
print ("Esta es mi carta: " , mi_carta )
v = mi_carta.get_valor
```


Método especial: **__str__**

Dado que es muy frecuente querer imprimir los atributos de un objeto, Python provee un método especial para retornar un string: **__str__** (sobrecarga de la función **str()**)

Al incluir el nombre de un objeto dentro de la instrucción **print()**, se invoca automáticamente el método **__str__**

Por ejemplo, en la clase **Carta** incluimos la definición del método **__str__(self)** :

```
def __str__(self):
    return str(self.valor) + self.pinta
```

Y luego podemos imprimir haciendo:

print ("Esta es mi carta: " **,** mi_carta **)**

También se invoca con:

str (mi_carta **)**

Ejemplo 2: Podemos crear ahora una clase **Jugador**

Los atributos de **Jugador** serán nombre y mano (lista de cartas) :

```
class Jugador:
    " Atributos: nombre, mano """
    def __init__(self, nombre):
        self.nombre = nombre
        self.mano = []

    def recibir_carta(self, carta):
        self.mano.append(carta)

    def __str__(self):
        m = ""
        for c in self.mano:
            m += str(c) + " "
        m += "-> " + str(self.contar_mano())
        return m
```

El método **recibir_carta** permite agregar cartas a la mano del Jugador.

En el método **__str__** queremos retornar un string que muestre las cartas que tenemos en la mano y nos de la suma de puntos. Falta definir el método **contar_mano**.

33

Ejemplo 2: Definamos el método **contar_mano** (dentro de **Jugador**)

Los atributos de **Jugador** serán nombre y mano (lista de cartas) :

```
class Jugador:
```

```
def contar_mano(self):   # calcular los puntos en la "mano"
    conteo = 0; ases = 0
    for carta in self.mano:
        conteo += carta.get_valor() # aquí consideramos los Ases como 1
        if carta.valor == 'A':
            ases += 1          # contamos los Ases
    while ases > 0 and conteo <= 11:
        conteo += 10     # porque un As vale 1 u 11
        ases -= 1
    return conteo
```

Si no nos pasamos de 21, queremos que el As valga 11. Si nos pasamos de 21, hacemos al As valer 1.

Veamos a continuación el programa completo para jugar 21 Real (BlackJack).

34

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Introducción a la Programación
IIC1103

Prof. Ignacio Casas
icasas@ing.puc.cl

Tema 09 – Programación Orientada a Objetos

Parte 1 Ejemplos

Colaboración de Mauricio Arriagada